REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
AUGUST 2011	Conference Paper – Post Print	May 2010 – October 2010

4. TITLE AND SUBTITLE

A MULTI-STEP SIMULATION APPROACH TOWARD SECURE FAULT TOLERANT SYSTEM EVALUATION

5a. CONTRACT NUMBER
In-House

5b. GRANT NUMBER
N/A

5c. PROGRAM ELEMENT NUMBER
62702F

6. AUTHOR(S)

Ruchika Mehresh, Shambhu Upadhyaya, and Kevin Kwiat

5d. PROJECT NUMBER
23G4

5e. TASK NUMBER
IH

5f. WORK UNIT NUMBER
01

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

AFRL/Information Directorate
Rome Research Site/RIGD
525 Brooks Road
Rome NY 13441

University at Buffalo
Dept of Computer Science and Engineering
Buffalo NY 14260

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Air Force Research Laboratory/Information Directorate
Rome Research Site
26 Electronic Parkway
Rome NY 13441

10. SPONSOR/MONITOR'S ACRONYM(S)
AFRL/RI

11. SPONSORING/MONITORING AGENCY REPORT NUMBER
AFRL-RI-RS-TP-2011-4

12. DISTRIBUTION AVAILABILITY STATEMENT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED. PA #: 88ABW-2010-3092.
Date Cleared: June 9, 2010.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

As new techniques of fault tolerance and security emerge, so does the need for suitable tools to evaluate them. This paper presents a multi-step, simulation-based performance evaluation methodology for secure fault tolerant systems. A divide-and-conquer approach is used to model the entire secure system in a way that allows the use of different analytical tools at different levels of granularity. This evaluation procedure tries to strike a balance between the efficiency, effort, cost and accuracy of a system's performance analysis. This approach is demonstrated in a step-by-step manner by analyzing the performance of a secure and fault tolerant system using a JAVA implementation in conjunction with the ARENA simulation.

15. SUBJECT TERMS

Architecture, Fault Tolerance, Modeling, Security, Simulation

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Kevin Kwiat
U	U	U	UU	6	19b. TELEPHONE NUMBER (*Include area code*) N/A

2010 29th IEEE International Symposium on Reliable Distributed Systems

A Multi-Step Simulation Approach Toward Secure Fault Tolerant System Evaluation[1]

Ruchika Mehresh and Shambhu J. Upadhyaya
Department of Computer Science and Engineering
State University of New York at Buffalo
Buffalo, New York 14260
{rmehresh, shambhu}@ buffalo.edu

Kevin Kwiat
Air Force Research Laboratory
525 Brooks Road
Rome, New York 13441-4505
kwiatk@rl.af.mil

***Abstract*- As new techniques of fault tolerance and security emerge, so does the need for suitable tools to evaluate them. Generally, the security of a system can be estimated and verified via logical test cases, but the performance overhead of security algorithms on a system needs to be numerically analyzed. The diversity in security methods and design of fault tolerant systems make it impossible for researchers to come up with a standard, affordable and openly available simulation tool, evaluation framework or an experimental test-bed. Therefore, researchers choose from a wide range of available modeling-based, implementation-based or simulation-based approaches in order to evaluate their designs. All of these approaches have certain merits and several drawbacks. For instance, development of a system prototype provides a more accurate system analysis but unlike simulation, it is not highly scalable. This paper presents a multi-step, simulation-based performance evaluation methodology for secure fault tolerant systems. We use a divide-and-conquer approach to model the entire secure system in a way that allows the use of different analytical tools at different levels of granularity. This evaluation procedure tries to strike a balance between the efficiency, effort, cost and accuracy of a system's performance analysis. We demonstrate this approach in a step-by-step manner by analyzing the performance of a secure and fault tolerant system using a JAVA implementation in conjunction with the ARENA simulation.**

Index Terms— Architecture, Fault Tolerance, Modeling, Security, Simulation

I. INTRODUCTION

The choice of the evaluation tool greatly affects the cost, efficiency and effort required to analyze a new secure fault tolerant design or idea. There are many tools available to the research community for such analysis but mostly their application areas are specialized. As a result, it is sometimes very inefficient to evaluate a new idea using the available tools. To modify an open source tool for the desired evaluation requires a lot of effort. Researchers may also choose to develop their own tools (theoretical models, simulations, etc.) but a wrong approach can affect the cost of development and accuracy of results adversely. For instance, in simulation the validity of results cannot be established unless the system model is thoroughly validated and verified.

There are generally three approaches to evaluate a system design based on its current state of development. When the system architecture is not established, researchers generally use CTMC (Continuous time Markov chains) models to analyze their systems. There have been several tools developed to solve the CMTC models [1]. In the second approach, when system design is available, simulation tools are used to model the functional behavior of the system. The third approach is based on conducting experimentation on a real-world system prototype to test the availability, dependability and reliability of the system. The drawback of relying completely on either one of them is the increase in complexity or cost, or decrease in the accuracy of results. If system implementation is chosen as the evaluation tool then the results will be more accurate and more representative of the real world conditions (like hardware faults, network conditions, etc.) However, not only the implementation is expensive to scale, but sometimes it may not even be possible or affordable to develop. In such a case, simulation may be preferred because it is easier to develop and it can scale very rapidly at low cost. Simulation enables the study of feasibility, behavior and performance without the need of an actual system. It can also run at any speed compared to the real world and thus can be used to test a wide range of scenarios in lesser time than with a real system. However, accuracy is a major issue with simulation models. Designing a highly accurate and valid simulation model is not only difficult but sometimes costly in terms of resource and time.

We propose a multi-step approach that can be widely used for evaluating secure fault tolerant systems. This approach involves a combination of theoretical analysis, pilot system implementation and simulation. This mix of analytical techniques can be optimized to obtain an evaluation procedure that minimizes its development effort and cost (resources and time), and maximizes its accuracy and efficiency.

To demonstrate this approach, we have evaluated a secure fault tolerant system that performs majority voting. The first step of this demonstration is a pilot system implementation in Java. The results obtained from this pilot implementation are used to parameterize the second step of the demonstration, which is a simplified system simulation.

Section II discusses the rationale for our proposed approach by surveying the various existing evaluation techniques. Section III discusses the proposed multi-step approach in detail. Section IV demonstrates the use of multi-step approach for performance analysis and reports the results. The paper concludes in Section V with a discussion on future work.

[1] Approved for Public Release; Distribution Unlimited: 88ABW-2010 3092, dated 9 June 2010

1060-9857/10 $26.00 © 2010 IEEE
DOI 10.1109/SRDS.2010.53

II. RATIONALE

Most of the feasibility, behavior and performance studies conducted in the literature for fault tolerant systems use theoretical analysis, actual system implementation, or the available simulation tools. One of the earliest attempts to develop fault tolerant systems, SIFT (Software implemented Fault Tolerance) [2] is completely software-based and uses loose synchronization of processors and memory. Since then, many tools and frameworks, like Chameleon [3], Globus [4], etc., have been proposed to develop and evaluate fault tolerant systems. Apache Hadoop [5] is a Java software framework that can be used to develop data-intensive, fault tolerant, distributed applications.

Many simulation tools and languages exist that can be used for developing and analyzing fault tolerant system models. CSIM [6] is a process-oriented, discrete-event simulation language that enables quick construction of concise system models. OMNeT++ [7] is a C++ simulation library and framework. Mobius [8] is a software tool used for modeling the behavior of complex systems. GridSim [9] is a grid simulation toolkit for resource modeling and application scheduling for parallel and distributed computing.

DEPEND [10] provides an integrated design and fault injection environment for system level dependability analysis. The fault injection can be defined as a stochastic process in DEPEND so it emulates a real world scenario. However, it is a specialized tool that concentrates majorly on fault injection and dependability analysis.

Security of a system can be generally verified using threat models and a series of logical test cases. However, the application of a security algorithm to a system results in performance overhead that must be within acceptable limits.

Simulation can only approximate the behavior of a real system. In a real system, the components (memory, processor speed, network bandwidth, etc.) have complex inputs, interconnections and dependencies that are not always easy to model. In addition to this, two similar components, such as two processors, can have different behaviors even if they are modeled as the same for the purpose of simulation. These factors introduce disparity between the results obtained from simulation and the results from experimentation. As discussed above, we need to verify if the performance overhead of a security application is within acceptable limits. For this purpose, we need the simulation to perform as closely to a real world system as possible. To reduce this disparity between the simulation and experimentation results, there exist many general-purpose simulation tools that allow the designing of stochastic system models. Stochastic parameters/variables can take into account a lot of unpredictable real-world factors. However, this approach presents the challenge of specifying the stochastic system parameters and variables, like probability distributions, seeds, etc. Mostly, values for defining the system parameters and variables are taken from prior projects, sometimes without proper justification or verification, or are simply assumed.

The differences between simulation and experimentation results can be ignored if only approximate comparisons are required (like observing a linear or exponential relationship between the input and output quantities). However, if the objective is to obtain the results as close to the experimentation results as possible (as required in our system because there may not be any existing results to compare), then we need to realistically parameterize our simulation model.

Mostly researchers validate their simulation design by comparing the simulation results (for lower scale values) with the results obtained from the system implementation. In many cases, the actual system may not exist and hence it is not possible to validate a simulation model. Hence, the need is to simplify the simulation model, so it can be easily verified for the logic. Adding excessive details to a model makes it more complex to understand and is prone to design errors. For instance, in designing a network application, an attempt to design the various time-variant factors that affect network performance will not only be impossible to precisely model, but will also increase the probability of design errors. So simulation designers generally make simplifying assumptions like the availability of a 100Mbps network bandwidth at all times. However, the application rarely gets to use the entire bandwidth. So the execution time obtained from a simulation for a network application will be much more optimistic than in a real world implementation. Designers generally go to a specific level of granularity in simulation designing and then start making assumptions beyond that level. Our proposed approach tries to realistically estimate these “assumed” values. This will provide more statistically accurate results along with a much simpler simulation model that will be less prone to design errors.

Another reason for proposing this multi-step approach is to deal with long or unbounded execution times for system implementations. Sometimes there is a system prototype available but the runtime is directly proportional to some parameter/variable, for instance, the workload size. In this scenario, if we need to run large workloads and one application run takes days, it becomes very inefficient to experiment with a large number of design alternatives. So, a system simulation for this problem will be a better solution. However, designing a realistic enough simulation is again the challenge here.

III. THE MULTI-STEP APPROACH

The proposed approach is a combination of three concepts: Modular decomposition, modular composability and parameterization [11]. Modular decomposition consists of breaking down a problem into smaller elements. Modular composition involves production of elements that can be freely combined with each other to provide new functionality. Parameterization is the process of defining the necessary parameters for the problem.

The multi-step approach consists of a modular functional model of the system. This model is hierarchical in terms of the level of detail/granularity. We start with the most abstract form of the model and work downwards toward a more detailed level/finer granularity. If a module can be further decomposed into sub-modules that satisfy the composability property, we move down further into the hierarchy. On reaching the level of maximum decomposition where the complexity of the module is very high, we replace it with a black-box instead of analyzing it further for decomposition. This *black-boxed level* of detail is complex to model for simulation purposes but it is

simple to be described using stochastic values (fault rate, bandwidth, etc.) that are statistically derived from real world experimentations. It can also be defined via theoretical analysis, like the use of queuing theory in case of scheduling.

The concept used by this approach is that the simulation model develops down the levels of granularity (becomes finer) and the development of system prototype/theoretical analysis goes up the levels of granularity. They meet in the middle where the accuracy, simplicity and efficiency of this approach can be maximized. Refer to Fig 1. Since the upper levels of this model design are simple, they are less prone to design errors.

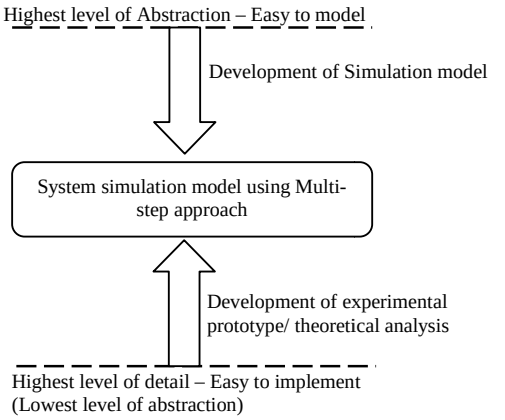

Figure 1: Development process of simulation model using multi-step approach

IV. Demonstration

To demonstrate the multi-step approach, we focus on a simple system that executes applications in a secure, fault tolerant environment. Execution of the workload is made secure and fault tolerant by running its parallel copies (or replicas) on several systems in lockstep. Application level checkpoints are taken at constant intervals and a recovery is initiated in case of a failure/security threat. In the end, the results from all the replicas are compared to get the final outcome. For the purpose of this demonstration, we will show how to model this system using the multi-step approach and measure the effect of failures/security breaches on the execution time of the workload. The assumption in this model is that all the security breaches will cause some kind of inconsistency in the checkpoint. A failure cannot be distinguished from a security breach except that a constant security threat is assumed to be permanent, while failures can be either transient or permanent.

Such systems are generally used as a basis for developing mission-assurance applications. These applications can run for hours or days. Therefore, a large number of re-runs for such systems for experimentation purposes is not very efficient. Thus arises the need for a simulation and the multi-step approach.

A. System Architecture

As shown in Fig. 2, the system's backbone consists of a coordinator and three replicas. Coordinator has three major components: a heartbeat manager, a checkpoint manager and a majority voter. The workload is fed to the coordinator and it then becomes the coordinator's job to execute this workload in a fault tolerant and secure environment to return a final and valid result. To accomplish this, coordinator generates several copies (replicas) of this workload and distributes them among the available servers over the network. The servers execute these replicas in lockstep by coordinating via centralized checkpointing. All the replicas send periodic heartbeats to the coordinator, which are handled by the heartbeat manager. When heartbeat manager does not receive a heartbeat from a replica over a specified amount of time, it assumes that the replica has become unavailable and the coordinator no more considers it as a part of the ongoing execution.

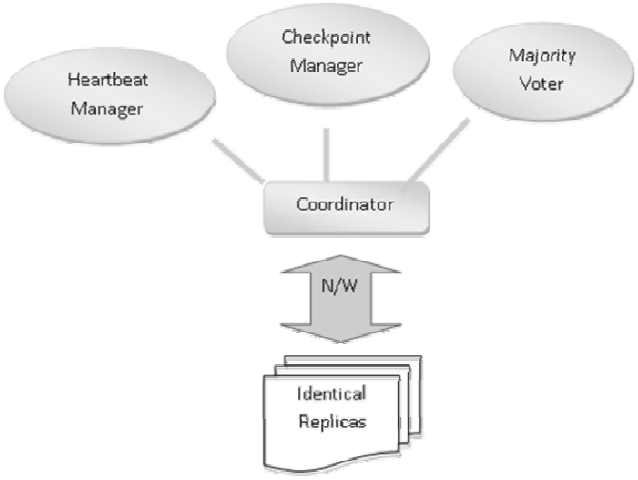

Figure 2: System Architecture

Checkpoints are the snapshots of the status of workload execution. The replicas are marked at specific points where the checkpoints are taken and sent periodically to the coordinator. Coordinator then compares the application-level checkpoints obtained from these replicas to make sure that the application state is consistent for all of them and there is no security breach at any of the replicas. If even one of the checkpoints is inconsistent with the rest, the coordinator signals for a rollback recovery and sends the last stored consistent checkpoint to all the replicas. Otherwise, it sends a confirmation to proceed. All the replicas stop and wait until they get the positive confirmation from the coordinator. Thus, all replicas follow a stop-and-go protocol [12].

In case of a rollback, replicas reset their respective states to the last consistent checkpoint sent to them by the coordinator. This way, all the replicas are again in the same consistent state and the lockstep execution is achieved. However, if a fault/security breach is permanent (or occurring too often) and the checkpoint from a system is never consistent with the other checkpoints, the rollbacks (and thus execution) can go on infinitely. Therefore, a threshold is specified on the number of rollbacks that can be initiated due to a particular replica. After a replica crosses this threshold, it is no more considered a part of the future decision making. When all the replicas have calculated the final results, they are sent to the coordinator for majority voting. Coordinator then delivers the final result to the user.

B. Modeling the System using the Multi-step Approach

As discussed above, this system needs to resort to simulation so that the experimentation can be completed relatively faster than the real world time.

At an abstract level the system is simple, but as the granularity becomes finer, a lot of complexities arise. We construct the system model piece-by-piece and where it gets complicated or unpredictable, we black-box that level of detail to save the effort in designing it. This black-box will be parameterized using the experimentation with the system prototype. Note that we do not have to develop/built the entire system for these parameters/values.

We chose Java for this implementation because of its easy-to-use API for programming socket communication and our level of familiarity with it. For simulation purposes, we chose to use discrete event simulation. Discrete event simulation is generally of three types—event-oriented, activity-oriented and process-oriented. We could choose any of these to model and simulate our system. However, the system architecture is such that process-oriented approach would be the most convenient and accurate one. Workload can be defined as an entity with attributes like size, arrival time, checkpoint rate, etc. The various stages of processing like network, replica execution, heartbeat management, etc. are modeled as separate processes. We have a variety of tools like CSIM, JavaSim, and ARENA that we could use to design this simulation. However, we chose Arena for this demonstration since it has a user friendly drag-and-drop interface for developing the simulation model [13].

At the highest level of abstraction, the three main modules that we need to consider are: Network, Coordinator and Replica. Refer to Fig. 3. Now we go through each module to see if it can be further decomposed into sub-modules. To verify that a new level of hierarchy can be defined, we need to investigate the potential sub-modules for the following two properties:

i) *Composability*: The functionalities of the potential sub-modules can be composed to provide the functionalities of their parent module.

ii) *Sufficiency*: The functionalities of the potential sub-modules collectively describe the entire set of functionalities of their parent module.

The first module 'Coordinator' has three sub-modules by design. These three sub-modules collectively describe the entire set of functionalities provided by the coordinator. Therefore, these three sub-modules are composedly sufficient to describe the coordinator. Hence, we move down one level of the hierarchy for the coordinator module.

The second module represents network that is unpredictable and is complicated to decompose further. Moreover, though we have a fixed network communication protocol but a real system cannot always strictly follow the protocol. For instance, a replica may skip sending a few heartbeats/ checkpoint due to a busy processor. So network can be modeled as a black-box, parameterized using the data collected from the experiments on the system prototype (implemented in Java).

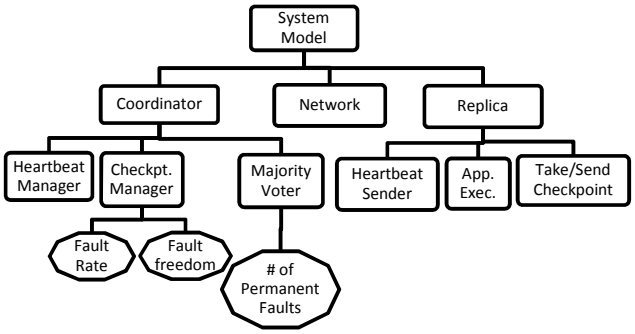

Figure 3: System model hierarchy of height 4 using the multi-step approach

Similarly, going down the levels of hierarchy, we get a tree of height 4 with the lowest level that cannot be further decomposed.

C. Parameterization

The data recorded from the several runs of experiments can be converted into probability distributions with the help of data analysis tools such as Minitab, Arena Input Analyzer, etc. These distributions can then parameterize the lowest level of our simulation model. We have used Arena [13] input analyzer for data analysis and Arena student version for simulation of this system. Arena input analyzer fits the best possible probability distribution to the data. Various tests (like Chi-square test) can be conducted using these tools to find out how well the selected distribution fits the data.

D. Results

Using this simulation we experimented with the workload size to determine its effect on the execution runtime in the presence of faults. As shown in Fig. 4, the runtime grows linearly with the increasing workload size in the presence of the fault percentage as shown in Table 1.

TABLE 1: DIFFERENT FAULT LEVELS FOR ANALYSIS IN FIGURE 4

	Replica 1	Replica 2	Replica 3
Time	0%	0%	0%
Time 1	10%	0%	0%
Time 2	25%	10%	10%

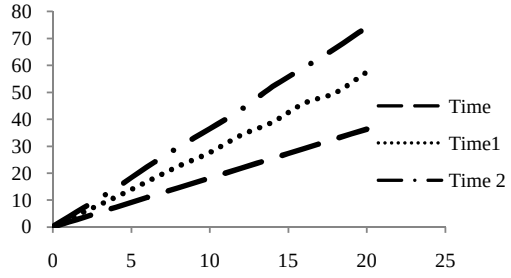

Figure 4: Simulation runtime (ms) Vs Workload size (MB)

We also experimented with a workload of fixed size (1,000 MB) with no threshold on the number of rollbacks that can be

initiated per replica. When the fault rate in just one of the replicas is increased, the runtime increases exponentially.

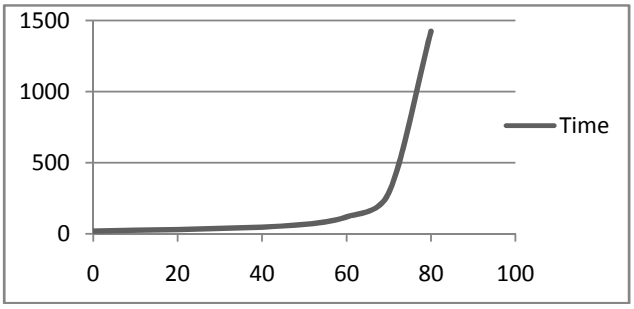

Figure 5: Simulation runtime (ms) Vs Fault rate for a fixed workload

As can be seen from Fig. 5, runtime explodes if the fault rate is very high in one of the replicas. For this purpose, designers cap the fault rate by ousting any replica that has more than a specified fault rate.

Multi-step approach also provides researchers with the independence to decide their desired level of simplicity versus accuracy. For instance, in our sample system, we could have defined all the modules at the first level of hierarchy as black-boxes. This would have simplified our simulation a lot, but it would not be as accurate.

V. Conclusion

In this work, we proposed an approach to reduce the effort and cost required to analyze a secure fault tolerant system and increase the accuracy of analysis. This approach aims at finding a balance between the theoretical analysis, implementation and simulation approaches to solve the problem of analyzing a system. It also enables researchers to develop their own project-specific analysis with higher confidence in its validity. We demonstrated the analysis of a secure, fault tolerant, centralized, replicated system using the proposed multi-step approach. This system was analyzed by using implementation in conjunction with simulation to evaluate the effect of faults on the execution time of the workload.

In the future, we intend to aid the development of problem-specific simulation tools by providing a simulation framework based on the multi-step approach. This simulation framework will assist the designers to develop the models using multi-step approach and give them the independence to choose the desired level of accuracy and efficiency. We will also apply this approach to evaluate a new secure proactive recovery paradigm that we are currently working on to address the survivability of mission-critical applications. Another plan is to investigate the applicability of this approach to other fields of research, such as wireless networks, cloud computing and evaluate their related security issues.

VI. Acknowledgement

This work was supported in part by U.S. Air Force Research Laboratory Grant No. 200821J.

References

[1] R. Geist and K. Trivedi, "Reliability Estimation of Fault-Tolerant Systems: Tools and Techniques", *Computer*, vol. 23, no. 7, pp. 52-61, July 1990

[2] J.H. Wensley, "SIFT Software Implemented Fault Tolerance", *Proc. Fall Joint Computer Conf., AFIPS, vol. 41, pp. 243-253, 1972*

[3] Zbigniew Kalbarczyk, Ravishankar K. Iyer, Saurabh Bagchi, Keith Whisnant, "Chameleon: A Software Infrastructure for Adaptive Fault Tolerance", *IEEE Transactions on Parallel and Distributed Systems Vol. 10, NO. 6, p.560-579, JUNE 1999*

[4] http://www.globus.org/

[5] http://hadoop.apache.org/

[6] Herb Schwetman, "CSIM: a C-based process-oriented simulation language", *Proceedings of the 18th conference on Winter simulation, p.387-396, December 08-10, 1986, Washington, D.C., United States*

[7] http://www.omnetpp.org/

[8] http://www.mobius.illinois.edu/

[9] http://www.gridbus.org/gridsim/

[10] Kumar K. Goswami , Ravishankar K. Iyer , Luke Young, "DEPEND: A Simulation-Based Environment for System Level Dependability Analysis", *IEEE Transactions on Computers, v.46 n.1, p.60-74, January 1997*

[11] B. Meyer, Object-Oriented Software Construction, *Prentice Hall, 1988*

[12] E. N. (Mootaz) Elnozahy , Lorenzo Alvisi , Yi-Min Wang , David B. Johnson, "A survey of rollback-recovery protocols in message-passing systems", *ACM Computing Surveys (CSUR), v.34 n.3, p.375-408, September 2002*

[13] http://www.arenasimulation.com/